# 1 Multiplicative Weights (MW)

*Note: These lecture notes are closely based on those of Aaron Roth [1].*

## 1.1 MultiplicativeWeights

**The Problem.** Imagine that we have $N$ experts and we are operating in the following regime

- In rounds $1, \ldots, T$, the algorithm chooses some expert $i^{(t)}$
- In round $t$, each expert $i$ experiences a loss $\ell_i^{(t)} \in [0, 1]$. The algorithm experiences the loss of the expert it chooses $\ell_A^{(t)} = \ell_{i^{(t)}}^{(t)}$

The total loss of expert $i$ is $L_i^{(T)} = \sum_{t=1}^{T} \ell_i^{(t)}$. The total loss of the algorithm is $L_A^T = \sum_{t=1}^{T} \ell_A^t$. The goal of the algorithm is to obtain loss not much worse than the best expert: $\min_i L_i^{(T)}$.

**The Algorithm.** The following algorithm solves the problem. It sets $\eta = \sqrt{\frac{\ln N}{T}}$.

**procedure** MULTIPLICATIVEWEIGHTS
  Set weights $w_i^{(1)} \leftarrow 1$ for all experts $i$
  **for** $t = 1$ to $T$ **do**
    $W^{(t)} \leftarrow \sum_{i=1}^{N} w_i^{(t)}$
    Choose expert $i$ with probability $w_i^{(t)}/W^{(t)}$
    For each $i$, $w_i^{(t+1)} \leftarrow w_i^{(t)}(1 - \eta \ell_i^{(t)})$
  **end for**
**end procedure**

**Analysis.** The following theorem shows MultiplicativeWeights does quite well.

**Theorem 1.** *Let $L_A^{(T)}$ be the loss of the MultiplicativeWeights algorithm, and $k$ be any expert. Then:*

$$\frac{1}{T} E(L_A^{(T)}) \leq \frac{1}{T} L_k^{(T)} + 2\sqrt{\frac{\ln N}{T}};$$

*which we get by setting $\eta = \sqrt{\frac{\ln N}{T}}$.*

**Proof:** Let $F^{(t)}$ be the expected loss of MultiplicativeWeights in round $t$. By linearity of expectation, we know $E(L_{MW}^{(T)}) = \sum_{t=1}^{T} F^{(t)}$. We also know that

$$F^{(t)} = \frac{\sum_{i=1}^{N} w_i^{(t)} \ell_i^{(t)}}{W^{(t)}}$$

So what is $W^{(t)}$? We know that $W^{(1)} = N$ and by the algorithm:

$$W^{(t+1)} = W^{(t)} - \sum_{i=1}^{N} \eta w_i^{(t)} \ell_i^{(t)} = W^{(t)}(1 - \eta F^{(t)}).$$

Solving this recurrence, we get

$$W^{(T+1)} = N \prod_{t=1}^{T} (1 - \eta F^{(t)}).$$

Taking the log, and using the fact that $\ln(1-x) \leq -x$, we get:

$$\begin{aligned} \ln W^{(T+1)} &= \ln N + \sum_{t=1}^{T} \ln(1 - \eta F^{(t)}) \\ &\leq \ln N - \eta \sum_{t=1}^{T} F^{(t)} \\ &= \ln N - \eta E(L_A^{(T)}) \end{aligned}$$

We next use the fact that $W^{(T+1)} \geq w_k^{(T+1)}$. And also the inequality $\ln(1-x) \geq -x - x^2$, which holds for $x \in [0, 1/2)$. To see this, note $\frac{1}{1-t} = 1 + t + t^2 + t^3 + \ldots$ for $|t| < 1$. Then $-\ln(1-x) = \int_0^x \frac{1}{1-t} dt$. So integrating the geometric sum equation above, we get $\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} + \ldots$, from which the inequality follows when $x \in [0, 1/2)$.

Using these two fact, we get for every $k$:

$$\begin{aligned} \ln W^{(T+1)} &\geq \ln w_k^{(T+1)} \\ &= \sum_{t=1}^{T} \ln(1 - \eta \ell_k^{(t)}) \\ &\geq -\sum_{t=1}^{T} \eta \ell_k^{(t)} - \sum_{t=1}^{T} (\eta \ell_k^{(t)})^2 \\ &\geq -\eta L_k^{(t)} - \eta^2 T \end{aligned}$$

Combining these two bounds, we get:

$$-\eta L_k^{(t)} - \eta^2 T \leq \ln N - \eta E(L_A^{(T)})$$

Isolating $E(L_A^{(T)})$ we get:

$$E(L_A^{(T)}) \leq -L_k^{(t)} - \eta T + (\ln N)/\eta$$

Dividing by $T$ and setting $\eta = \sqrt{\frac{\ln N}{T}}$ gives the promised result. □

## 1.2 MWVectorBounded

**The Algorithm.** The following algorithm is competitive with any convex combination of experts.

**procedure** MWVectorBounded
  Set weights $w_i^{(1)} \leftarrow 1$ for all experts $i$
  **for** $t = 1$ to $T$ **do**
    $W^{(t)} \leftarrow \sum_{i=1}^{N} w_i^{(t)}$
    Play vector $w^{(t)}$, where $w^{(t)}[i] = w_i^{(t)}/W^{(t)}$ for $i \in [1, N]$
    For each $i$, $w_i^{(t+1)} \leftarrow w_i^{(t)}(1 - \eta \ell_i^{(t)})$
  **end for**
**end procedure**

**The Analysis.** Recall that $\Delta(N)$ is the set of all length $N$ vectors having all entries non-negative and summing to 1. We have the following.

**Theorem 2.** *Setting $\eta = \sqrt{\frac{\ln N}{T}}$, for any sequence of losses $\ell^{(t)} \in [0,1]^N$, MWVectorBounded gets:*

$$\frac{1}{T}\sum_{t=1}^{T}\langle w^{(t)}, \ell^{(t)}\rangle \leq \min_{w^* \in \Delta(N)} \frac{1}{T}\sum_{t=1}^{T}\langle w^*, \ell^{(t)}\rangle + 2\sqrt{\frac{\ln N}{T}}$$

**Proof:** The left hand side is exactly the expected loss of MultiplicativeWeights. Similarly, the "loss of expert $i$" corresponds to $\sum_{t=1}^{T}\langle e_i, \ell^{(t)}\rangle$, where $e_i$ is the standard basis vector with a 1 in position $i$ and 0's elsewhere. Finally, observe that for every sequence of possible losses:

$$\min_{w^* \in \Delta(N)} \sum_{t=1}^{T}\langle w^*, \ell^{(t)}\rangle = \sum_{t=1}^{T}\langle e_{i^*}, \ell^{(t)}\rangle$$

where $i^* = \arg\min_{i \in [1,N]} \sum_{t=1}^{T} \ell_i^{(T)}$. So minimizing regret to the best basis vector $e_{i*}$, i.e. the best expert, minimizes regret to the best vector $w^* \in \Delta(N)$. Thus, the result follows by Theorem 1. □

### 1.3 Arbitrary Losses and Vector Sums

Let $B_N(R_1)$ be the set of $n$ dimensional vectors with nonnegative entries that sum up to at most $R_1$. Below are changes we make to MWVector in order to compete with vectors in $B_N(R_1)$ and handle losses in the range $[-R_2/2, R_2/2]$. Again we will set $\eta = \sqrt{\frac{\ln N}{T}}$.

**procedure** MWVECTOR $(R_1, R_2)$
  Add a $N+1$ expert, with $\ell_{N+1}^{(t)} = 0$ for all $t$.
  Set $u_i^{(1)} \leftarrow 1$ for all experts $i$
  **for** $t = 1$ to $T$ **do**
    $U^{(t)} \leftarrow \sum_{i=1} u_i^{(t)}$
    Play the vector $w^{(t)}$, where $w^{(t)}[i] = R_1 u_i^{(t)}/U^{(t)}$ for all $i$
    For each $i$, $u_i^{(t+1)} \leftarrow u_i^{(t)}(1 - \eta(\ell_i^{(t)} + R_2/2)/R_2))$
  **end for**
**end procedure**

The algorithm makes the following changes compared to MWVectorBounded

- **Weights:** A $N+1$ expert is added that always has loss of 0. The algorithm uses "internal" weights $u_i$ and outputs "external" weights $w_i$ that are now scaled by $R_1$.
- **Losses:** We first add $R_2/2$ to every loss value, and then divide the new loss value by $R_2$. This gives normalized loss values all in $[0,1]$.

**Theorem 3.** *MWVector ensures for any $w^* \in B_N(R_1)$ that:*

$$\frac{1}{T}\sum_{t=1}^{T}\langle w^{(t)}, \ell^{(t)}\rangle \leq \min_{w^* \in B_N(R_1)} \frac{1}{T}\sum_{t=1}^{T}\langle w^*, \ell^{(t)}\rangle + 2R_1R_2\sqrt{\frac{\ln N}{T}}$$

**Proof:** First, note that the $N+1$ expert allows for the normalized, "internal" weight values, $u_i^{(t)}/U^{(t)}$, to sum to anything in the range $[0,1]$. When we scale these values by $R_1$, we thus obtain any vector with positive entries that sum to anything less than $R_1$. Second, note that adding $R_2/2$ to each of the losses does not change the "regret", i.e. the value $\left\langle w^{(t)}, \ell^{(t)} \right\rangle - \left\langle w^*, \ell^{(t)} \right\rangle$.

Finally, if we divide the coordinates of $w_t$ by $R_1$ and the coordinates of $\ell_t$ by $R_2$, we get, by Theorem 2, that:

$$\frac{1}{T}\sum_{t=1}^{T}\left\langle \frac{w^{(t)}}{R_1}, \frac{\ell^{(t)}}{R_2} \right\rangle \leq \min_{w^* \in B_N(R_1)} \frac{1}{T}\sum_{t=1}^{T}\left\langle \frac{w^*}{R_1}, \frac{\ell^{(t)}}{R_2} \right\rangle + 2\sqrt{\frac{\ln N}{T}}.$$

Multiplying the above inequality by $R_1 R_2$ gives the promised inequality. □

# 2 Winnow

## 2.1 Problem

Assume we are given feature vectors, $a_1, \ldots, a_m$, each having a classification of either $-1$ or $+1$. We want to find a hyperplane that correctly separates the points.

For simplicity, we can define $a_j$ to be $s_j a_j$, where $s_j \in \{-1, +1\}$ is the classification of example $j$. Then, we want to find vector $x$, such that for each attribute vector $i$, $x \cdot a_i \geq 0$. Without loss of generality, we can assume that $x$ is in $\Delta(n)$ where $n$ is length of each attribute vector. [1]

**Example.** We are given three feature vectors: $(2,-1)$, class $-1$; $(-4,1)$, class 1, $(-10,1)$, classs $-1$. Multiplying each vector by its class gives: $(-2,1)$, $(-4,1)$, $(10,-1)$. Then, we see that $x = (1/10, 9/10)$ is one vector that correctly classifies all of these.

## 2.2 Algorithm.

Let $N$ be the number of attributes in each feature vector, i.e. the length of each feature vector. Let $M$ be the maximum value of any entry in any feature vector.

**procedure** WINNOW
Run MWVector ($R_1 = 1, R_2 = 2M$) with the following settings:
**Experts:** Associate an expert with each of the $N$ attributes.
**Losses:** In round $t$, find some $a_j$, where $\langle w^{(t)}, a_j \rangle < 0$. Then set $\ell_i^{(t)} \leftarrow -a_j[i]$.
When $\langle w^{(t)}, a_j \rangle \geq 0$ for all $a_j$, terminate.
**end procedure**

## 2.3 Analysis

Suppose there is a vector $v^* \in B_N(1)$ such that for all $j$, $\langle v^*, a_j \rangle \geq \epsilon$, for some fixed $\epsilon > 0$. Then, we have the following result.

**Theorem 4.** *Winnow terminates in at most* $\frac{16M^2 \ln N}{\epsilon^2}$ *rounds.*

[1] Why? We can simulate negative values in $x$ by adding in attributes that are the negative of all the old attributes; this only doubles the number of attributes.

**Proof:** Let $T$ be the round in which Winnow terminates. Then, for all $t < T$, $\langle w^{(t)}, \ell^{(t)} \rangle > 0$. Thus, we have

$$\begin{aligned} 0 &< \sum_{t=1}^{T} \langle w^{(t)}, \ell^{(t)} \rangle \\ &\leq \sum_{t=1}^{T} \langle w^*, \ell^{(t)} \rangle + 4M\sqrt{T \ln N} \\ &\leq -T\epsilon + 4M\sqrt{T \ln N} \end{aligned}$$

In the above, the second step holds by Theorem 3. The third step holds by noting that $\sum_{t=1}^{T} \langle w^*, \ell^{(t)} \rangle \leq \sum_{t=1}^{T} \langle v^*, \ell^{(t)} \rangle \leq \epsilon$. Isolating $T$ in the above inequality gives:

$$T \leq \frac{16M^2 \ln N}{\epsilon^2}$$

□

# 3 Minimax Theorem

## 3.1 Game Theory

Two-players, *Min* and *Max*, are playing a game characterized by a $m$ by $n$ matrix $C$. Player Min chooses an action from the set $A_1 = \{1, \ldots, m\}$, and player Max chooses an action from the set $A_2 = \{1, \ldots, n\}$. The cost of the game is given by the function $C$: $A_1 \times A_2 \to \mathbb{R}$.[2]

A *mixed* strategy for Min is a probability distribution over $A_1$: $p \in \Delta(m)$; a mixed strategy for Max is a a probability distribution over $A_2$: $q \in \Delta(n)$. Then the *expected cost* of the game is

$$C(p, q) = \sum_{i=1}^{m} \sum_{j=1}^{n} C[i, j] p_i q_j = q^T C p.$$

One of the players may play a *pure* strategy - i.e. does not randomize. For example, Max might always play strategy $y \in [n]$. Then we write:

$$C(p, y) = \sum_{i=1}^{m} C[i, y] p_i = (e_y)^T C p.$$

**Example.** A example (zero-sum) game is given by the following 3 by 3 payoff matrix $C$.

| | Rock | Paper | Scissors |
|---|---|---|---|
| Rock | 0 | -1 | 1 |
| Paper | -1 | 0 | 1 |
| Scissors | 1 | -1 | 0 |

[2]This type of game where the costs of both players are given by a single matrix is called zero-sum

**MaxMin and MinMax.** Normally, games are played simultaneously, but what if Min has to announce her (mixed) strategy to Max first? Or vice versa? We can calculate the cost of the game in each case as follows:

$$\max\min(C) = \max_{q\in\Delta(n)} \min_{x\in[m]} \sum_{j=1}^{n} q_i C[x,i]$$

$$\min\max(C) = \min_{p\in\Delta(m)} \max_{y\in[n]} \sum_{i=1}^{m} p_i C[i,y]$$

Note that the player who plays second always just chooses a single action - their *best response* - since there's no need to randomize if you already know your opponents probability distribution.

### 3.2 Minimax Theorem

It turns out that it doesn't matter which player goes first. In particular, we will show that $\min\max(C) = \max\min(C)$ for all matrices $C$. We will eventually show this using our Multiplicative Weights algorithm, but first a definition.
Vectors $p \in \Delta(m)$ and $q \in \Delta(n)$ form an *$\epsilon$-approximate minimax equilibrium* for a game $C$ iff:

$$\max_{y\in[n]} C(p,y) - \epsilon \le C(p,q) \le \min_{x\in[m]} C(x,q) + \epsilon$$

The following algorithm will help us obtain such an $\epsilon$-approximate equilibrium over an $m$ by $n$ matrix $C$, where all entries in $C$ are normalized to be in the range $[0,1]$

**procedure** ComputeEQ $(C, \epsilon)$
  $T \leftarrow \frac{4\log m}{\epsilon^2}$
  Initialize a copy of MWVectorBounded to run over $w^{(t)} \in \Delta(m)$
  **for** $t = 1$ to $T$ **do**
    $y^{(t)} \leftarrow \arg\max_{y\in[n]} C(w^{(t)}, y)$
    $\ell_i^{(t)} \leftarrow C[i, y^{(t)}]$ for all $i \in [m]$
    Pass $\ell^{(t)}$ to the MWVectorBounded algorithm
  **end for**
  Return: $(\bar{x}, \bar{y}) \leftarrow \left(\frac{1}{T}\sum_{t=1}^{T} w^{(t)}, \frac{1}{T}\sum_{t=1}^{T} e_{y^{(t)}}\right)$
**end procedure**

**Theorem 5.** *For any $\epsilon > 0$ and any game $C$, ComputeEQ returns vectors $(\bar{x}, \bar{y})$ that are an $\epsilon$-approximate minimax equilibrium.*

**Proof:** First, observe that for any $x \in \Delta(m)$,

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T} C(x, y^{(t)}) &= C\left(x, \frac{1}{T}\sum_{t=1}^{T} e_{y^{(t)}}\right) \\ &= C(x, \bar{y})\end{aligned}$$

Similarly, for any $y \in \Delta(n)$, $\frac{1}{T}\sum_{t=1}^{T} C(w^{(t)}, y) = C(\bar{x}, y)$. These are both true since linearity of the $C$ function allows to push sums and scalars inside it.

Let $x^* = \arg\min_x C(x, \bar{y})$ and $y^* = \arg\max_y C(\bar{x}, y)$. Then, MWVectorBounded ensures, by Theorem 2:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T} C(w^{(t)}, y^{(t)}) &\leq \frac{1}{T}\sum_{t=1}^{T} C(x^*, y^{(t)}) + \sqrt{\frac{4\log m}{T}} \\ &= C(x^*, \bar{y}) + \sqrt{\frac{4\log m}{T}}\end{aligned}$$

Also, by choice of $y^{(t)}$ in algorithm ComputeEQ:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T} C(w^{(t)}, y^{(t)}) &\geq \frac{1}{T}\sum_{t=1}^{T} C(w^{(t)}, y^*) \\ &= C(\bar{x}, y^*)\end{aligned}$$

Putting these two inequalities together, we get that:

$$C(\bar{x}, y^*) \leq C(x^*, \bar{y}) + \sqrt{\frac{4\log m}{T}}$$

Adding and subtracting $C(\bar{x}, \bar{y})$ and rearranging, we get:

$$(C(\bar{x}, \bar{y}) - C(x^*, \bar{y})) + (C(\bar{x}, y^*) - C(\bar{x}, \bar{y})) \leq \sqrt{\frac{4\log m}{T}}$$

By definition of $x^*$ and $y^*$, both terms on the LHS are non-negative. Hence, both:

$$C(\bar{x}, \bar{y}) - C(x^*, \bar{y}) \leq \sqrt{\frac{4\log m}{T}} \qquad C(\bar{x}, y^*) - C(\bar{x}, \bar{y}) \leq \sqrt{\frac{4\log m}{T}}$$

This implies that $(\bar{x}, \bar{y})$ forms a $\sqrt{\frac{4\log m}{T}}$-approximate minimax equilibrium. By our choice of $T$, $\sqrt{\frac{4\log m}{T}} \leq \epsilon$. □

Now, Von Neumann's minimax theorem follows as a corollary.

**Theorem 6.** *(Von Neumann Minimax) For any zero-sum game $C$,*

$$\min\max(C) = \max\min(C)$$

**Proof:** By Theorem 5, for any game $C$ and $\epsilon > 0$, we can find $\bar{x}$, $\bar{y}$ such that:

$$\min\max(C) - \epsilon \leq \max_{y\in[n]} C(\bar{x}, y) - \epsilon \leq C(\bar{x}, \bar{y}) \leq \min_{x\in[m]} C(x, \bar{y}) + \epsilon \leq \max\min(C) + \epsilon$$

Thus, $\min\max(C) \leq \max\min(C) + 2\epsilon$ for all $\epsilon > 0$, so $\min\max(C) \leq \max\min(C)$. Since player Min can only do worse when she releases her strategy first, we also know that $\min\max(C) \geq \max\min(C)$. Thus, the two quantities are equal! □

# 4 Boosting

Define a *labelled datapoint* to be a pair $(x, y) \in X \times Y$, where $X$ is some space of features and $Y$ is some space of labels. Frequently, $X = \mathbb{R}^d$ and $Y = \{0, 1\}$. A *data set* is a collection of labelled datapoints.

Our goal is to find a function $f : X \to Y$ for predicting labels from features with high accuracy. In particular, given a predictor $f : X \to Y$, the accuracy of $f$ on a data set $D$ of $n$ items is:

$$acc(f, D) = \frac{1}{n} \sum_{i=1}^{n} I(f(x_i) = y_i)$$

We can also define weighted prediction accuracy for any weighting $w \in \Delta(n)$ of the $n$ points:

$$acc(f, D, w) = \frac{1}{n} \sum_{i=1}^{n} w_i I(f(x_i) = y_i)$$

In the above $I(p)$ is an indicator function which is 1 if the predicate $p$ is true and 0 otherwise

**Hypothesis, Learners, and Weak and Strong Learners** A *hypothesis class* $H$ is a collection of hypothesis $h$ that map from $X$ to $Y$. A *weighted learning algorithm* $A$ with range $H$ is a mapping from datasets and weight vectors to hypothesis in $H$.

$A$ is a *weak learning algorithm* for $D$ if for every distribution $w \in \Delta(n)$, $A(D, w) = h$ such that:

$$acc(h, D, w) \geq .51$$

$A$ is a *strong learning algorithm* for $D$ if $A(D) = h$ such that:

$$acc(h, D) = 1$$

## 4.1 Weak Learner to Strong Learner

Remarkably, we can use a weak learning algorithm for a dataset $D$ to obtain a strong learning algorithm.

**Theorem 7.** *For any dataset $D$, if there exists a polytime weak learning algorithm for $D$, then there exists a polytime strong learning algorithm for $D$.*

**Proof:** Our approach will use game theory. Let $H$ be the hypothesis class used by the weak learning algorithm $A$. Define a zero-sum game as follows:

- The action space for the Min player ("DATA") is the set of data points: $A_1 = D$.
- The action space for the Max player ("LEARNER") is the $A_2 = H$
- The cost function is defined as: $C((x_i, y_i), h) = I(h(x_i) = y_i))$

Since $A$ is a weak learner for $D$, we can compute the min max value of the game. In particular, this is how well the player LEARNER can do if the player DATA must go first:

$$\begin{aligned}
\min\max(C) &= \min_{w\in\Delta(n)} \max_{h\in H} \sum_{i=1}^{n} w_i C((x_i, y_i), h) \\
&= \min_{w\in\Delta(n)} \max_{h\in H} \sum_{i=1}^{n} w_i I((h(x_i) = y_i) \\
&= \min_{w\in\Delta(n)} \max_{h\in H} \sum_{i=1}^{n} acc(h, D, w) \\
&\geq .51
\end{aligned}$$

The last line holds since the player LEARNER can use the weak learner algorithm $A$. But, then we also know:

$$\begin{aligned}
\min\max(C) &= \max\min(C) \\
&= \max_{p\in\Delta(|H|)} \min_{i\in[n]} \sum_{h\in H} p_h I(h(x_i) = y_i) \\
&\geq .51
\end{aligned}$$

The first step follows by the minimax theorem (Theorem 6), since we know player LEARNER can do just as well if she goes second. In the second step, the value $p$ chosen in $\Delta(H)$ is the mixed strategy for player LEARNER. The third step follows by our lowerbound on min max above.

Thus, there is some fixed distribution $p^*$ over hypothesis $h \in H$ such that at least .51 of the hypothesis weighted by $p$ correctly label **every** point $(x_i, y_i)$. Consider the following "weighted majority vote" classifier:

$$f_{p^*}(x) = \arg\max_{y\in Y} \sum_{h\in H} p^* I(h(x) = y)$$

From our minimax calculations above, we know that for *any* $(x_i, y_i) \in D$, we have:

$$\sum_{h\in H} p_h I(h(x_i) = y_i) \quad = \sum_{h\in H: h(x_i)=y_i} p^*_h \quad \geq \quad .51$$

Thus, by construction, $f_{p^*}(x_i) = y_i$. □

## 4.2 Some Details and Resource Analysis

Note that algorithm ComputeEQ can only compute an $\epsilon$-equilibrium, not an exact one. But that's not a problem so long as we set $\epsilon$ small enough so that payoff of the game is still $> .5$. For example, $\epsilon = .001$ works fine since $.51 - .001 > .5$ holds.

### 4.2.1 Algorithm

So what does the final algorithm look like if we "inline" the call to ComputeEQ? A direct version of our algorithm Boost, for dataset $D$, and weak-learning algorithm $A$ is given below. Recall that the weak-learning algorithm $A$ can, for any weighting $w$ of data set $D$, always find a hypothesis $h \in H$ that correctly classifies at least a .51 weighted fraction of the items in $D$. Finally, let $n = |D|$ and $m = |H|$.

In the following algorithm, the weight vector, $w^{(t)}$, gives the strategy for player DATA, and the (weighted) $h^{(t)}$ values give the strategy for the player LEARNER.

**procedure** BOOST $(D, A)$
  $T \leftarrow \frac{4 \log n}{\epsilon^2}$
  Initialize a copy of MWVectorBounded to run over $w^{(t)} \in \Delta(n)$
  **for** $t = 1$ to $T$ **do**
    $h^{(t)} \leftarrow A(D, w^{(t)})$
    $\ell_i^{(t)} \leftarrow I(h^{(t)}(x_i) = y_i)$ for all $i \in [m]$
    Pass $\ell^{(t)}$ to the MWVectorBounded algorithm
  **end for**
  $\bar{p} \leftarrow \frac{1}{T} \sum_{t=1}^{T} e_{h^{(t)}}$
  Return: $f_{\bar{p}}(x)$, where $f_{\bar{p}}(x) = \arg\max_{y \in Y} \sum_{h \in H} \bar{p}_h I(h(x) = y)$
**end procedure**

### 4.2.2 Analysis

How long does our algorithm Boost run for? Since $\epsilon$ is a constant, on a dataset of size $n$, it runs for $O(\log n)$ iterations. In each round, it makes a single call to the weak learning algorithm $A$. Then it updates the weight distributions over the $n$ data points in $D$, which takes $O(n)$.

Note that the function $f_{\bar{p}}(x)$ is concisely representable since $\bar{p}$ has support over only the $T = O(\log n)$ hypothesis, $h^{(t)}$, chosen at the start of each iteration.

Finally, note that if the weak learner returns "simple" functions such as linear classifiers or depth-2 decision trees, then the function $f_{\bar{p}}(x)$ will also not be too complex. In particular, it can be described as the average of no more than $O(\log n)$ of these simple functions. This simplicity is a positive indication that $f_{\bar{p}}(x)$ will generalize well, performing well not just on the training data, but also on the test data.

## 5 Linear Programming

Let's end with a "simple" application of MW to linear programming. Imagine we have a linear program of the form:

Maximize: $\sum_{i=1}^{n} c_i x_i$

Subject to each constraint $j \in [d]$:

$$\sum_{i=1}^{n} a_{i,j} x_i \leq b_j$$

We can solve this using MW, provided that we can find violated constraints whenever they exist. Note that there could even be an exponential number of constraints, provided we can find a violated constraint quickly, if any constraint is violated!

Our approach is first to convert our linear program into a *linear feasible* problem, where there is no maximize function. Instead, if we know the solution has value OPT, we put in the constraint:

$$\sum_{i=1}^{n} -c_i x_i \leq -OPT$$

Since we don't know OPT in practice, we can find it via binary search. Henceforth, we focus only on linear feasible LPs.

### 5.1 The Algorithm

The idea is to run MW where the "experts" are the $n$ variables in the LP, and the weights will eventually give settings of the variables that approximately satisfy all $d$ constraints. In particular, the algorithm will be given a parameter $\alpha > 0$, and we will require the output values must violate each constraint by no more than $\alpha$.

**procedure** SolveLP $(\{a, b\}_{j\in[d]},\ R_1,\ R_2,\ \alpha)$
  Initialize MWVector to produce vectors $w \in B_n(R_1)$
  **while** there is a constraint $j^{(t)}$ such that $\sum_{i=1}^{n} a_{i,j} w_i^{(t)} \geq b_j + \alpha$ **do**
    $\ell_i^{(t)} \leftarrow a_{i,j^{(t)}}$ for all $i \in [n]$
    Pass $\ell^{(t)}$ to MWVector
    $t \leftarrow t + 1$ and $w_{(t)}$ is the updated weight vector.
  **end while**
  Output: $x = w^{(t)}$
**end procedure**

### 5.2 Analysis

We first give a definition, and then our main theorem.

**Definition 1.** *Given a linear feasibility problem: $(\{a, b\}_{j\in[d]}$, vector $x$ is an $\alpha$-* feasible solution *if for all $j \in [d]$,*

$$\sum_{i=1}^{n} a_{i,j} x_i \leq b_j + \alpha$$

**Theorem 8.** *Let $(\{a, b\}_{j\in[d]}$ be a linear feasibility problem with a feasible solution $x^* \in B_n(R_1)$, and such that $\max_{i\in[n],j\in[d]} |a_{i,j}| \leq R_2/2$. Then, SolveLP when run on that LP problem returns an $\alpha$-feasible solution in a number of iterations at most:*

$$T \leq \frac{4R_1^2 R_2^2 \ln n}{\alpha^2}$$

**Proof:** If the algorithm terminates, it outputs a $\alpha$-feasible solution by definition. How many iterations before it terminates? Let $x^*$ be the feasible solution. By definition of $x^*$, in iteration $t$, we know $\sum_{i=1}^{n} x_i^* \ell_i^{(t)} \leq b_{j^{(t)}}$. By definition of the algorithm, in iteration $t$, we know $\sum_{i=1}^{n} x_i^{(t)} \ell_i^{(t)} \leq b_{j^{(t)}}$

Hence, the regret of the algorithm is at least:

$$\frac{1}{T}\sum_{t=1}^{T} \langle w^{(t)}, \ell^{(t)} \rangle - \frac{1}{T}\sum_{t=1}^{T} \langle w^{(*)}, \ell^{(t)} \geq \frac{1}{T}\sum_{t=1}^{T} b_{j^{(t)}} + \alpha - b_{j^{(t)}} \geq \alpha.$$

But, by Theorem 3,

$$\frac{1}{T}\sum_{t=1}^{T} \langle w^{(t)}, \ell^{(t)} \rangle - \frac{1}{T}\sum_{t=1}^{T} \langle w^{(*)}, \ell^{(t)} \leq 2R_1 R_2 \sqrt{\frac{\ln n}{T}}$$

Hence, we must have:

$$2R_1 R_2 \sqrt{\frac{\ln n}{T}} \geq \alpha$$

Solving for $T$ gives the theorem. □

## References

[1] Aaron Roth. Polynomial Weights, 2021. https://www.cis.upenn.edu/~aaroth/courses/MWlectures.pdf.